Composition of Context-Free Motion Description Languages

Wenqi Zhang and Herbert G. Tanner

UD MEEG Technical Report
Number MEEG TR-2010-0002
August 2010

Department of Mechanical Engineering
University of Delaware
Newark, Delaware
URL: http://www.me.udel.edu

Composition of Context-Free Motion Description Languages

Wenqi Zhang and Herbert G. Tanner

University of Delaware, Department of Mechanical Engineering
MEEG TR-2010-0002

August 2010

Abstract

We introduce a new formalism to define compositions of interacting heterogeneous systems, described by extended motion description languages (MDLes). The novelty of the formalism is in producing a composed system with a behavior that could be a superset of the union of the behaviors of its generators. We prove closedness of MDLes under this composition and we indicate that in the class of systems modeled using MDLes, language equivalence is decidable. Our approach consists of representing MDLes as normed processes, recursively defined as a guarded system of recursion equations in restricted Greibach Normal Form over a basic process algebra. Basic processes have well defined semantics for composition, which we exploit to establish the properties of our composed MDLes.

1 Introduction

As robotic systems are called to perform increasingly complex tasks, control designers are resorting more frequently to switching or hybrid control algorithms. A need arises for a methodology to "stitch" together different controlled robotic behaviors in a way that a final objective is ultimately met. The task of deciding on the sequence of behaviors currently falls on the control designer. In fact, it is quite natural for humans to develop plans that solve complex problems. It is not so for machines.

Automating the process of planning has traditionally been a problem in the realm of artificial intelligence, and the complexity issues involved in discrete planning are well known [1]. It has also been recognized that there is a gap between such discrete high level planning algorithms, and low level motion control implementation (see [2] and the references therein). The approach in [2] attempts to bridge this gap using linear temporal logic (LTL) to describe the task to be performed in a formalism similar to natural language, and then translate the logic formula into an automaton that gives rise to a hybrid controller.

The approach followed in this paper is along similar lines, but it is rather "bottom-up." We start by assuming the existence of a set of designed behavioral primitive, and we proceed by developing a framework that dictates how these behaviors are sequentially synthesized into plans that drive the system into a desired state. In that sense our behavioral primitive form an alphabet of actions. We need to emphasize that neither [2] nor this paper suggest another *programming* language, such as any one of the high level languages that are currently being used to implement robot controllers and drive hardware. In fact, this paper attempts to utilize an existing motion control "meta-language," [3, 4] to abstract low level controllers —implemented in any possible programming language— into elementary behaviors in a way to facilitate high level planning.

Motion Description Languages (MDLs) [5] translate collections of control algorithms into robust and reusable software [3]. MDLe is an extension of the early definitions of motion description languages [6]. It is a device-independent programming language for hybrid motion control, which allows one to compose complex, interrupt-driven control laws from a set of simple primitives, and a number of syntactic rules [3, 4]. MDLes (e standing for "extended") have been criticized for not capturing interaction between systems. This paper is an attempt to address this issue, and set a framework in which MDLes can be composed, verified, and allow automated motion and task planning for collections of heterogeneous robotic systems.

We identify MDLes as recursive systems in some basic process algebra (BPA) written in Greibach Normal Form (Lemma 1). We propose a simple context-free grammar that generates MDLes and then we use the machinery available for BPAs to formally define a composition operation for MDLes at the level of grammars. The technical core of this paper indicates how appropriately defined MDLe grammars can be composed (Definition 8), and language equivalence (whether two such grammars generate the same finite traces), is decidable up to bisimulation. The main difference of our composition operation is the appearance in the composed system of events (transitions) not enabled in the generators: the composed system can behave in ways its generators cannot. In our approach, one still needs to identify beforehand these events that can be activated after the composition. But the proposed definition partially captures the fact that the whole can be more than the sum of its parts.

Our definition of MDLe composition, viewed independently, is not based on the BPA technical machinery. However, by identifying MDLes as a subclass of BPAs we are able to borrow the syntax and semantics of the BPA merge operator (instead of defining a new MDLE operator), and thus establish closeness and decidability properties for MDLE compositions.

Are basic process algebras a good formalism to map hybrid robotic systems to discrete models of computation? It is a formalism out of many possible. Our justification for choosing BPAs comes first from our desire to model robotic systems using MDLes. In [7, 8], it is shown that deterministic pushdown automata are decidable up to bisimulation equivalence. In this paper, we arrive at an equivalent statement by exploiting BPA properties.

Another modeling formulation is that of maneuver automata [9], which are finite automata that produce sequences of predetermined maneuvers for unmanned vehicles. Admissible motion is expressed as the set of traces the automaton accepts. Maneuver automata, however, generate regular languages, a set that does not include MDLes [4].

Concurrent systems can also be expressed as petri nets [10]. Petri nets generate context-sensitive languages. They are therefore more expressive than BPAs but this comes at a cost: bisimulation is undecidable for petri nets [11], which poses an obstacle for further analysis and abstraction. MDLes, on the other hand, are context-free [12]. The tools we use to arrive at this decidability result are the properties of BPAs introduced in [13, 14, 15], and refined in [16].

Although other modeling tools may be available, we feel that BPAs strike a reasonable balance between complexity and expressiveness when it comes to modeling systems expressed by, and controlled through, MDLes. Showing that under the extended notion of composition we introduce, the resulting system is an MDLe (Lemma 3), and that the decidability properties are preserved (Corollary 2), gives us hope that the resulting (big) system can be abstracted to the point that some of the available model checkers [17, 18, 19] can be used to construct admissible motion plans in the form of "counterexamples."

In Section 2 that follows we provide some introductory technical material on motion description languages (the interested reader is referred to [4] and [20]). Section 3 serves as a brief introduction to basic process algebras. Our contribution is contained in Section 4: there we show that MDLes are a subset of BPAs, we define MDLe composition, we show that composition is closed, preserves bisimilarity, and indicate why language equivalence in MDLes is decidable. Section 5 presents an example that illustrates how MDLes are composed to produce systems with behaviors not existent in the component systems. Section 6 summarizes the paper.

2 MDLe Preliminaries

2.1 Definitions

Every MDLe string consists of a control part, an interrupt part, and the special symbols ")", "(", and ",". A dynamical system can generally be described in the form

$$\dot{x} = f(x,u), \quad y = h(x); \quad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m,\ y \in \mathbb{R}^p, \tag{1}$$

where x is the state of the system; u is the control input; and y is the measurable output. Let U be a finite set of feedback control laws (or quarks [12]) $u : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$, for (1), and B a finite set of boolean functions $\xi : \mathbb{R}^p \times \mathbb{R} \to \{0,1\}$ of output y and time $t \leq T \in \mathbb{R}_+$ (the interrupt quarks [12]).

The basic element of an MDLe is an *atom,* denoted (u,ξ), where u is a control law selected from U, and ξ is the *interrupt* selected from set B. To *evaluate* or *run* an atom (u,ξ), means to apply the input u to (1) until the interrupt function ξ evaluates true ($\xi = 1$). An MDLe *plan* is composed of a sequence of atoms. For example, evaluating the plan $a = ((u_1,\xi_1),(u_2,\xi_2))$ means that the system state x, flows along $\dot{x} = f(x,u_1)$ until $\xi_1 = 1$, and then along $\dot{x} = f(x,u_2)$ until $\xi_2 = 1$. Plans can also be composed to generate higher order strings, as in $b = ((u_3,\xi_3),a,(u_4,\xi_4))$.

2.2 MDLes are context-free

The pumping lemma is utilized in [12] to show that an MDLe is not a regular language; rather, it is context-free. Context-free languages are generated by context-free grammars (CFGs), which can always be expressed in Chomsky normal form. A variation of the Chomsky normal form, is the Greibach normal form.

Definition 1 ([16]). *A context-free grammar in which every production rule is of the form $A \to a\alpha$, where A is a variable, a is a terminal, and α is a possibly empty string of variables, is said to be in* Greibach normal form *(*GNF*). If, moreover, the length of α (in symbols) does not exceed 2, we say that the context-free grammar is in restricted Greibach normal form.*

3 Basic Process Algebras

3.1 Definitions

A BPA is essentially a mathematical structure consisted of set of constants, $A = \{a, b, c, \dots\}$, called *atomic actions*, a set Σ_{BPA} of two binary operators on these constants (the alternative composition $+$ and the sequential composition $\cdot$), and a set of axioms E_{BPA} that determines the properties of the operations on the atomic actions [16]. When the set of atomic actions A is assumed known, a basic process algebra is denoted simply as a couple in the form $\mathrm{BPA} = (\Sigma_{\mathrm{BPA}}, E_{\mathrm{BPA}})$. The set Σ_{BPA} is called *signature*, while set E_{BPA} *equation* set (hence the symbols). The theory associated to a BPA is considered to be parameterized by the set A, which is specified according to the particular application.

The symbol $\cdot$ denoting sequential composition is typically omitted, and we usually write xy instead of $x \cdot y$. We assume that $\cdot$ binds stronger than $+$, thus $(xy) + z = xy + z$ (brackets omitted). The set E_{BPA} consists of five axioms (or equations), appearing in Table 1. Composing atomic actions according to Table 1, yields more complex *processes*.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5

Table 1: The axioms of a BPA.

Any such process, is an element of some algebra satisfying the axioms of BPA, and all processes produced in this way make up the set P. The axiom system of Table 1 is the core of a variety of more extensive process axiomatizations:

- $x \cdot y$ is the process that first executes x, and upon completion of x, process y starts.
- $x + y$ is the process that either executes x, or executes y (but not both).

Just as in the case of finite state machines, processes are identified by the set of action sequences they admit. Some [21] prefer to include a set Atom of atomic processes or *atoms*. The set Proc of *processes* contains all terms in the free algebra over Atom generated by sequential composition and disjunction. Then a process algebra is defined by a finite set Π of productions of the form $X \xrightarrow{a} P$, where $X \in \mathrm{Atom}$, $a \in A$, and $P \in \mathrm{Proc}$. The semantics of the above production is as follows: atomic process X performs action a and evolves into process P. Let us identify a process with an automaton, in which a transition denotes the execution of an atomic action. The states of this automaton are all the processes derived through the set of production rules. Action relations are presented in Table 2, in which $x \xrightarrow{a} y$, with x and y being processes and a an atomic action, means that process x evolves into process y after the atomic action a is executed.

The symbol $\surd$ stands for successful termination. It is said that a relation is true if and only if it can be derived from the relations of Table 2. Note the distinction between the relation operator ($\to$) and sequential composition ($\cdot$): the fact that $x \xrightarrow{a} y$ does not imply that $y = x \cdot a$, since a is an action executed as x runs, not after it is completed. The only thing that can be inferred about action a is that it is an action that process x can execute.

$a \xrightarrow{a} \surd$	R1
$a \xrightarrow{a} x' \Rightarrow x+y \xrightarrow{a} x'$ and $y+x \xrightarrow{a} x'$	R2
$x \xrightarrow{a} \surd \Rightarrow x+y \xrightarrow{a} \surd$ and $y+x \xrightarrow{a} \surd$	R3
$x \xrightarrow{a} x' \Rightarrow xy \xrightarrow{a} x'y$	R4
$x \xrightarrow{a} \surd \Rightarrow xy \xrightarrow{a} y$	R5

Table 2: The operational semantics of BPA.

3.2 Recursive and guarded BPAs

Let us focus on a special type of BPAs with slightly finer semantics. The additional properties of this type of systems enable us to define composition more comfortably, and establish the decidability of language equivalence for the systems produced by means of composition.

Definition 2 ([22]). *A* recursive equation *over a* BPA *is an equation of the form $X = s(x)$, where X is a variable that can take values in P and $s(x)$ is a term over the* BPA *containing X, but no other variable.*

A set of recursive equations give rise to a specification:

Definition 3 ([22]). *A recursive specification E over a* BPA *is a set of recursion equations over the* BPA.

We thus have a set of variables $V = \{x_0, \cdots, x_n\}$, and equations of the form $X = s_x(V)$ with $x \in V$, where s_x is a term over the BPA containing variables in V. Set V contains one distinguished variable called the root variable x_0. A variable in V is called guarded in a given term, if it is preceded by an atomic action:

Definition 4 ([22]). *Let s be a term over a* BPA*, containing variable X.*

- *An occurrence of X in s is said to be* guarded*, if s has a sub-term of the form $a \cdot t$, where a is an atomic action, and t a term containing this occurrence of X; otherwise this occurrence of X in s is said to be* unguarded.
- *A term s is* completely guarded *if all occurrences of all variables in s are guarded. A recursive specification E is completely guarded if all right hand sides of all equations of E are completely guarded terms.*

Just as production rules can be thought to be in Greibach normal form, so can equations over a BPA.

Definition 5 ([22]). *If a system E of recursion equations is guarded and without brackets, then each recursion equation is of the form $X_i = \sum_j a_j \cdot \alpha_j$, where α_j is a possibly empty product (sequential composition) of atoms and variables. Now if, in addition, α_j is exclusively a product of variables, E is said to be in* Greibach normal form*, analogous to the same definition for context-free grammars. If each α_j in E has length not exceeding* 2*, E is in* restricted *Greibach normal form.*

3.3 Composition and bisimulation of BPAs

BPAs can be equipped with a merge operator, $\|$. Process $x\|y$ is the process that executes process x and y in parallel. Notice that we do not assert that the first action has terminated when the second one starts; this can depend on the implementation of a process. The left merge operator, $\lfloor\!\lfloor$, describes two processes that occur in parallel, in a way similar to $\|$, but with the restriction that the first step must come from the process on the left of the expression. With the new operators, the BPA axioms are expanded as shown in Table 3, and the action relations are enriched as shown in Table 4.

Two BPA processes p_1 and p_2 are *bisimilar*, if whenever p_1 performs a certain action, p_2 can perform the same action, and *vise versa*. The following definition of bisimulation equivalence for processes is quoted from [23], and is chosen only because of its conceptual association to similar definitions of bisimulation for transition systems, that have appeared in the controls literature [24].

Definition 6 ([23]). *A binary relation $\approx$ on the set of processes* Proc *is a bisimulation, if the following conditions are satisfied:*

$x+y=y+x$	A1
$(x+y)+z=x+(y+z)$	A2
$x+x=x$	A3
$(x+y)z=xz+yz$	A4
$(xy)z=x(yz)$	A5
$x\|y=x⌊\!⌊y+y⌊\!⌊x$	M1
$a⌊\!⌊x=ax$	M2
$ax⌊\!⌊y=a(x\|y)$	M3
$(x+y)⌊\!⌊z=x⌊\!⌊z+y⌊\!⌊z$	M4

Table 3: The BPA axioms, expanded with the introduction of merge ($\|$) and left merge ($⌊\!⌊$) operators.

$a\xrightarrow{a}\surd$	R1
$a\xrightarrow{a}x'\Rightarrow x+y\xrightarrow{a}x'$ and $y+x\xrightarrow{a}x'$	R2
$x\xrightarrow{a}\surd\Rightarrow x+y\xrightarrow{a}\surd$ and $y+x\xrightarrow{a}\surd$	R3
$x\xrightarrow{a}x'\Rightarrow xy\xrightarrow{a}x'y$	R4
$x\xrightarrow{a}\surd\Rightarrow xy\xrightarrow{a}y$	R5
$x\xrightarrow{a}x'\Rightarrow x\|y\xrightarrow{a}x'\|y$ and $y\|x\xrightarrow{a}y\|x'$	R6
$x\xrightarrow{a}\surd\Rightarrow x\|y\xrightarrow{a}y$ and $y\|x\xrightarrow{a}y$	R7
$x\xrightarrow{a}x'\Rightarrow x⌊\!⌊y\xrightarrow{a}x'\|y$	R8
$x\xrightarrow{a}\surd\Rightarrow x⌊\!⌊y\xrightarrow{a}y$	R9

Table 4: The action relations of BPA, expanded using the composition operators.

- *for all p, q, and p' in* Proc*, and $a\in A$ such that $p\approx q$ and $p\xrightarrow{a}p'$, there exists $q'\in$* Proc *such that $q\xrightarrow{a}q'$ and $p'\approx q'$.*
- *for all p, q, and q' in* Proc*, and $a\in A$ such that $p\approx q$ and $q\xrightarrow{a}q'$, there exists $p'\in$* Proc *such that $p\xrightarrow{a}p'$ and $q'\approx p'$.*

4 Main Results

4.1 MDLes are a special class of BPAs

The representation of an MDLe as a BPA requires an intermediate step, which is the expression of the former as a context-free grammar. We define a context-free grammar $G=(N,\eta,R,S)$ so that it generates a motion description language MDLe $=\{(u,\zeta):u\in U,\zeta\in B\}$, in the following way [12]:

- N is the finite set of non-terminal symbols E, where E is a valid *variable*;
- $\eta=\{u_i,\xi_i,(,),,\}$ is the finite set of terminals, which are the atoms of L,
- S is the start symbol in N;
- R is the rules by which we create MDLe strings:

$$S\rightarrow E \tag{2a}$$
$$E\rightarrow EE \tag{2b}$$
$$E\rightarrow (u_i,\xi_i) \tag{2c}$$
$$E\rightarrow (E,\xi_i) \tag{2d}$$
$$E\rightarrow \emptyset \tag{2e}$$

Rule 2d is called "encapsulation" [12], which is essentially a while-structure, and gives MDLe its context-free character. We now define the push-down automaton that is equivalent to the context-free grammar described above, as in [25]. Definition 7 allows us to conveniently switch between representations.

Definition 7. $P=(N,\eta,\Sigma,\Gamma,\delta,S,Z_0)$ *where*

- *N is the set of* states*, defined the same as* variables *in G;*
- *$\eta=\{u_i,\xi_i,(,),,\}$ is the set of* enabled events*, associated with transitions in P;*
- *$\Sigma=N\cup\eta$ is the stack alphabet;*

- $\Gamma : E \to \Gamma(E)$ *is the active event function;*
- $\delta : E \times \eta \to E$ *is the transition function,* $\delta(x,E) = y$ *means that there is a transition labeled by event* η *from state x to y;*
- $S \in N$ *is the start state, defined the same as the start state in G;*
- Z_0 *is the start symbol in stack;*

The range of Γ defines all active events, the ones that correspond to transitions the automaton can autonomously take. Note the distinction between η and $\Gamma(E)$: this is what enables us to capture actions the system cannot execute autonomously, but potentially can in collaboration with another system. We allow $\Gamma(E) \not\subseteq \eta$, but the transitions which the automaton can autonomously take are in $\Gamma(E) \cap \eta$. The next Lemma confines MDLes to set of languages generated by a special class of context-free grammars (CFGs).

Lemma 1. *An* MDL*e is produced by a* CFG *in Greibach normal form.*

Proof. We rewrite (2) in Chomsky normal form, an intermediate stage before we arriving at the Greibach normal form. Rewriting (2) in Chomsky normal form involves a sequence of steps, in which a transformation rule is applied to the set of rules written on the left to result in the rule set depicted on the right. Let us first combine rules (2) into a single one, using the disjunction operator |, for compactness. Then we give the resulting set of rules after each transformation.

$$\begin{array}{l} E \to EE \\ E \to (u_i, \xi_i) \\ E \to (E, \xi_i) \\ E \to \emptyset \end{array} \qquad\qquad E \to EE|(u_i,\xi_i)|(E,\xi_i)|\emptyset.$$

Step 1: Define a new start symbol S_0 to replace S.

$$\begin{array}{l} S \to E \\ E \to EE|(u_i,\xi_i)|(E,\xi_i)|\emptyset \end{array}$$

$$\begin{array}{l} S_0 \to S \\ S \to E \\ E \to EE|(u_i,\xi_i)|(E,\xi_i)|\emptyset \end{array}$$

Step 2: Remove $\emptyset$ from the rules that involve variable E.

$$\begin{array}{l} S_0 \to S \\ S \to E \\ E \to EE|(u_i,\xi_i)|(E,\xi_i)|\emptyset \end{array}$$

$$\begin{array}{l} S_0 \to S \\ S \to E|EE|(u_i,\xi_i)|(E,\xi_i) \\ E \to EE|(u_i,\xi_i)|(E,\xi_i) \end{array}$$

Step 3: Eliminate the original start variable S.

$$\begin{array}{l} S_0 \to S \\ S \to E \\ E \to EE|(u_i,\xi_i)|(E,\xi_i) \end{array}$$

$$\begin{array}{l} S_0 \to E|EE|(u_i,\xi_i)|(E,\xi_i) \\ E \to EE|(u_i,\xi_i)|(E,\xi_i) \end{array}$$

Step 4: Eliminate the unit rules.

$$S_0 \to E|EE|(u_i,\xi_i)|(E,\xi_i)$$
$$E \to EE|(u_i,\xi_i)|(E,\xi_i)$$

$$S_0 \to EE|(u_i,\xi_i)|(E,\xi_i)$$
$$E \to EE|(u_i,\xi_i)|(E,\xi_i)$$

Step 5: Convert the remaining rules into the proper form by adding variables and rules.

$$S_0 \to EE|(u_i,\xi_i)|(E,\xi_i)$$
$$E \to EE|(u_i,\xi_i)|(E,\xi_i)$$

$$S_0 \to EE|Lu_i,\xi_iR$$
$$|LE,\xi_iR$$
$$E \to EE|Lu_i,\xi_iR$$
$$|LE,\xi_iR$$
$$L \to ($$
$$R \to)$$

$$S_0 \to EE|Lu_i,B|A,B$$
$$E \to EE|Lu_i,B|A,B$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR$$

$$S_0 \to EE|D,B|A,B$$
$$E \to EE|D,B|A,B$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR$$
$$D \to Lu_i$$

$$S_0 \to EE|DF|AF$$
$$E \to EE|DF|AF$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR$$
$$D \to Lu_i$$
$$F \to ,B$$

$$S_0 \to EE$$
$$E \to EE|DF|AF$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR \qquad (3)$$
$$D \to LK$$
$$F \to ,B$$
$$H \to ,$$
$$J \to \xi_i$$
$$K \to u_i$$

Then we translate (3) into Greibach normal form, by first eliminating left-recursion.

Step 1: Add a new rule $T \to E|ET$ to eliminate left-recursion $E \to EE$.

$$S_0 \to EE|DF|AF$$
$$E \to EE|DF|AF$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR$$
$$D \to LK$$
$$F \to ,B$$
$$H \to ,$$
$$J \to \xi_i$$
$$K \to u_i$$

$$S_0 \to EE|DF|AF$$
$$|DFT|AFT$$
$$E \to DF|AF|DFT|AFT$$
$$L \to ($$
$$R \to)$$
$$A \to LE$$
$$B \to \xi_iR$$
$$D \to LK$$
$$F \to ,B$$
$$H \to ,$$
$$J \to \xi_i$$
$$K \to u_i$$
$$T \to E|ET$$

Step 2: The next step is to make all the other rules start with a terminal.

$$
\begin{aligned}
&S_0 \rightarrow EE|DF|AF \\
&|DFT|AFT \\
&E \rightarrow DF|AF \\
&|DFT|AFT \\
&L \rightarrow (\\
&R \rightarrow) \\
&A \rightarrow LE \\
&B \rightarrow \xi_i R \\
&D \rightarrow LK \\
&F \rightarrow ,B \\
&H \rightarrow , \\
&J \rightarrow \xi_i \\
&K \rightarrow u_i \\
&T \rightarrow E|ET
\end{aligned}
\qquad
\begin{aligned}
&S_0 \rightarrow (KFT \\
&|(EFT|(KF|(EF \\
&E \rightarrow (KF|(EF \\
&|(KFT|(EFT \\
&L \rightarrow (\\
&R \rightarrow) \\
&A \rightarrow (E \\
&B \rightarrow \xi_i R \\
&D \rightarrow (K \\
&F \rightarrow ,B \\
&H \rightarrow , \\
&J \rightarrow \xi_i \\
&K \rightarrow u_i \\
&T \rightarrow (KF|(EF \\
&|(KFT|(EFT
\end{aligned}
$$

Step 3: The final step is to convert all the rules in restricted GNF by adding rules.

$$
\begin{aligned}
&S_0 \rightarrow (KFT \\
&|(EFT|(KF|(EF \\
&E \rightarrow (KF|(EF \\
&|(KFT|(EFT \\
&L \rightarrow (\\
&R \rightarrow) \\
&A \rightarrow (E \\
&B \rightarrow \xi_i R \\
&D \rightarrow (K \\
&F \rightarrow ,B \\
&H \rightarrow , \\
&J \rightarrow \xi_i \\
&K \rightarrow u_i \\
&T \rightarrow (KF \\
&|(EF|(KFT|(EFT
\end{aligned}
\qquad
\begin{aligned}
&M \rightarrow KF \\
&N \rightarrow EF \\
&S_0 \rightarrow (MT|(NT \\
&|(KF|(EF \\
&E \rightarrow (M|(N|(MT|(NT \\
&L \rightarrow (\\
&R \rightarrow) \\
&A \rightarrow (E \\
&B \rightarrow \xi_i R \\
&D \rightarrow (K \\
&F \rightarrow ,B \\
&H \rightarrow , \\
&J \rightarrow \xi_i \\
&K \rightarrow u_i \\
&T \rightarrow (M|(N|(MT|(NT
\end{aligned}
$$

$$
\begin{aligned}
&M \rightarrow u_i F \\
&N \rightarrow (MF|(NF \\
&S_0 \rightarrow (MT|(NT|(KF|(EF \\
&E \rightarrow (M|(N|(MT|(NT \\
&L \rightarrow (\\
&R \rightarrow) \\
&A \rightarrow (E \\
&B \rightarrow \xi_i R \\
&D \rightarrow (K \\
&F \rightarrow ,B \\
&H \rightarrow , \\
&J \rightarrow \xi_i \\
&K \rightarrow u_i \\
&T \rightarrow (M|(N|(MT|(NT
\end{aligned}
\tag{4}
$$

□

The next Lemma states that an MDLe can be translated into a BPA in Greibach normal form [22].

Lemma 2. *The terms of an* MDLe *are a finite trace set of a normed process p, recursively defined by means of a guarded system of recursion equations in restricted Greibach normal form over a* BPA.

Proof. Lemma 1 allows us to express an MDLe as a CFG in Greibach normal form, which in addition satisfies the conditions of Notation 4.5 of [22]. We apply Notation 4.5 in conjunction with Proposition 5.2 of [22] to write the CFG of (4) as a BPA as follows.

- If S is the system represented as a CFG in Greibach normal form, let S' denote the system represented in BPA by replacing $|$ by $+$, and $\rightarrow$ by $=$.
- Let S' be in restricted Greibach normal form over the BPA, with unique solution p. Then $\mathrm{ftr}(p)$ (the set of finite traces of p) is just the context-free language generated by S.

Applying the change of notation suggested,

$$\begin{array}{l} M \rightarrow u_i F \\ N \rightarrow (MF|(NF \\ S_0 \rightarrow (MT|(NT|(KF|(EF \\ E \rightarrow (M|(N|(MT|(NT \\ L \rightarrow (\\ R \rightarrow) \\ A \rightarrow (E \\ B \rightarrow \xi_i R \\ D \rightarrow (K \\ F \rightarrow ,B \\ H \rightarrow , \\ J \rightarrow \xi_i \\ K \rightarrow u_i \\ T \rightarrow (M|(N|(MT|(NT \end{array} \qquad \begin{array}{l} M = u_i F \\ N = (MF + (NF \\ S_0 = (MT + (NT + \\ (KF + (EF \\ E = (M + (N + \\ (MT + (NT \\ L = (\\ R =) \\ A = (E \\ B = \xi_i R \\ D = (K \\ F = ,B \\ H = , \\ J = \xi_i \\ K = u_i \\ T = (M + (N + \\ (MT + (NT \end{array} \tag{5}$$

and thus we have a BPA in restricted Greibach normal form. Note that according to Definition 5, each variable string in the right hand side of (5) has length of at most two. By applying Proposition 5.2 of [22], to remove the parts of the system that do not contribute to the generation of the finite traces, we conclude that the BPA of (5) generates the strings of the original MDLe. □

4.2 Composition of MDLes

In the preceding section we distinguished between events associated to transitions a push-down automaton representing an MDLe can take autonomously, and events that cannot initiate transitions. Among the latter, there can be events that when *synchronized* with some of another push-down automaton (synchronization here implies a common interrupt function), become active and do initiate transitions. Given two push-down automata P_1 and P_2 defined according to Definition 7, we define the set $H \subseteq \eta_1 \cup \eta_2$ as the collection of events on which P_1 and P_2 should be synchronized. Set H includes those events that become active as a result of the composition of P_1 with P_2. Set H is composed of three components:

1. $(\Gamma_2 \cup \eta_1) \setminus (\Gamma_2 \cup \eta_2) \setminus (\Gamma_1 \cup \eta_1)$, (part I in Fig. 1), which contains enabled events of P_1 that P_1 can now activate because of P_2;
2. $(\Gamma_1 \cup \eta_2) \setminus (\Gamma_2 \cup \eta_2) \setminus (\Gamma_1 \cup \eta_1)$, (part III in Fig. 1), which contains enabled events of P_2 that now become active because of P_1; and
3. $(\Gamma_1 \cup \eta_2) \cap (\Gamma_2 \cup \eta_1)$, (part II in Fig. 1), which includes common active events in both systems.

Note that the components of H defined in 1 and 2 do not appear in the set of (active) events of the composed system under the conventional definition of composition [26]. Our definition of composition is stated as follows.

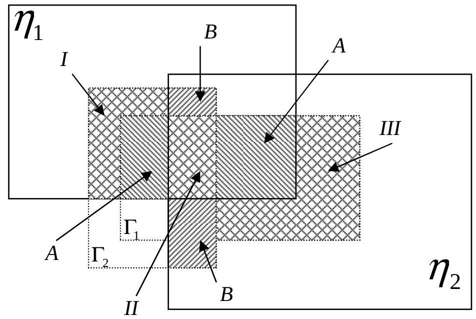

Figure 1: Enabled, active and common events. Set A includes private active events of P_1; set B contains private active events of P_2; sets I, II, and III represent the common active events of the composed system, the ones that make up H.

Definition 8. *Consider two* MDL*es, expressed as context-free grammars* $G_1 = (N_1, \eta_1, R, S_{01})$ *and* $G_2 = (N_2, \eta_2, R, S_{02})$*, both with rule sets R of the form* (2)*. Let* S_1 *and* S_2 *be their corresponding representations as a system of guarded recursive equations, in restricted Greibach normal form over a* BPA*. The composition of* G_1 *and* G_2 *is defined as the context-free grammar* $G = (N, \eta, R, S_0)$*, where*

- $N = N_1 \times N_2$*;*
- $\eta = \eta_{(1\|2)} = \eta_1 \cup \eta_2$*, which is the set of enabled events;*
- $S_0 = S_{01} \times S_{02}$*;*
- $$R(N \times \eta) := \begin{cases} (R(N_1,\eta), R(N_2,\eta)) & \textit{if } \eta \in H, \\ (R(N_1,\eta), N_2) & \textit{if } \eta \in A, \\ (N_1, R(N_2,\eta)) & \textit{if } \eta \in B \\ \textit{undefined}, & \textit{otherwise} \end{cases}$$

The transitions of the composed system still respect the grammar rules (2), however, the composition restricts the domain of R. The push-down automaton representing the composed system can be defined as follows:

Definition 9. $P_1 \| P_2 = (N, \eta_{(1\|2)}, \Sigma, \Gamma_{(1\|2)}, \delta, N_0, Z_0)$*, and*

- $N = N_1 \times N_2$*;*
- $\eta = \eta_{(1\|2)} = \eta_1 \cup \eta_2$
- $\Sigma = (N_1 \times N_2) \cup \eta_{(1\|2)}$ *the stack;*
- $\Gamma_{(1\|2)} = \Gamma_1 \cup \Gamma_2$
- $$\delta(N \times \eta) := \begin{cases} (R(N_1,\eta), R(N_2,\eta)) & \textit{if } \eta \in H, \\ (R(N_1,\eta), N_2) & \textit{if } \eta \in A, \\ (N_1, R(N_2,\eta)) & \textit{if } \eta \in B \\ \textit{undefined}, & \textit{otherwise} \end{cases}$$
- $N_0 = (N_{01} \times N_{02})$ *start state;*
- $Z_0 = (Z_{01} \times Z_{02})$ *start symbol in stack;*

For the composition of Definition 8 to be well defined, we need to make sure that when we compose variables and terminals of two systems in (guarded) Greibach normal form over a BPA, the result is a term that conforms to the same rules. This is established in the section that follows.

4.3 MDLes are closed under composition

The next result establishes that operation $\lfloor\!\lfloor$ is closed.

Lemma 3. *An* MDL*e written as a system of guarded recursive equations in restricted Greibach normal form is closed under the left merge $\lfloor\!\lfloor$ operator.*

Proof. Assume that G is written as a system of guarded recursive equations in restricted Greibach form, according to (4). We prove the claim by taking all merge combinations of variables in this representation, and showing that the result is a system of equations that are also guarded in restricted Greibach normal form. To simplify the proof, we will group the variables which have similar form.

$$\begin{array}{l} P = aX + aY + aXZ + aYZ = \{E, T, S_0\} \\ Q = aXW + aYW = \{N\} \\ \Omega = bG = \{M, A, B, D, F\} \\ \Pi = c = \{L, R, H, J, K\} \end{array} \tag{6}$$

According to Table 3,

$$\begin{array}{l} \Pi \lfloor\!\lfloor \Omega = c \lfloor\!\lfloor \Omega \overset{M2}{=} c\Omega \\ \Pi \lfloor\!\lfloor Q = c \lfloor\!\lfloor Q \overset{M2}{=} cQ \\ \Pi \lfloor\!\lfloor P = c \lfloor\!\lfloor P \overset{M2}{=} cP \\ \Omega \lfloor\!\lfloor Q = bG \lfloor\!\lfloor Q \overset{M3}{=} b(G \lfloor\!\lfloor Q) \\ \Omega \lfloor\!\lfloor P = bG \lfloor\!\lfloor P \overset{M3}{=} b(G \lfloor\!\lfloor P) \\ Q \lfloor\!\lfloor P = (aXW + aYW) \lfloor\!\lfloor P \overset{M4}{=} aXW \lfloor\!\lfloor P + aYW \lfloor\!\lfloor P \\ \quad \overset{M3}{=} a(XW \lfloor\!\lfloor P) + a(YW \lfloor\!\lfloor P) \end{array}$$

Note that reversing the order of variables in the above merge operations yields the same type of expressions encountered above:

$$
\begin{aligned}
P \mathbin{\lfloor\!\lfloor} Q &= (aX + aY + aXZ + aYZ) \mathbin{\lfloor\!\lfloor} Q \\
&\overset{M4}{=} aX \mathbin{\lfloor\!\lfloor} Q + aY \mathbin{\lfloor\!\lfloor} Q + aXZ \mathbin{\lfloor\!\lfloor} Q + aYZ \mathbin{\lfloor\!\lfloor} Q \\
&\overset{M3}{=} a(X \mathbin{\lfloor\!\lfloor} Q) + a(Y \mathbin{\lfloor\!\lfloor} Q) + a(XZ \mathbin{\lfloor\!\lfloor} Q) + a(YZ \mathbin{\lfloor\!\lfloor} Q) \\
P \mathbin{\lfloor\!\lfloor} \Omega &= (aX + aY + aXZ + aYZ) \mathbin{\lfloor\!\lfloor} \Omega \\
&\overset{M4}{=} aX \mathbin{\lfloor\!\lfloor} \Omega + aY \mathbin{\lfloor\!\lfloor} \Omega + aXZ \mathbin{\lfloor\!\lfloor} \Omega + aYZ \mathbin{\lfloor\!\lfloor} \Omega \\
&\overset{M3}{=} a(X \mathbin{\lfloor\!\lfloor} \Omega) + a(Y \mathbin{\lfloor\!\lfloor} \Omega) + a(XZ \mathbin{\lfloor\!\lfloor} \Omega) + a(YZ \mathbin{\lfloor\!\lfloor} \Omega) \\
P \mathbin{\lfloor\!\lfloor} \Pi &= (aX + aY + aXZ + aYZ) \mathbin{\lfloor\!\lfloor} \Pi \\
&\overset{M4}{=} aX \mathbin{\lfloor\!\lfloor} \Pi + aY \mathbin{\lfloor\!\lfloor} \Pi + aXZ \mathbin{\lfloor\!\lfloor} \Pi + aYZ \mathbin{\lfloor\!\lfloor} \Pi \\
&\overset{M3}{=} a(X \mathbin{\lfloor\!\lfloor} \Pi) + a(Y \mathbin{\lfloor\!\lfloor} \Pi) + a(XZ \mathbin{\lfloor\!\lfloor} \Pi) + a(YZ \mathbin{\lfloor\!\lfloor} \Pi) \\
Q \mathbin{\lfloor\!\lfloor} \Omega &= (aXW + aYW) \mathbin{\lfloor\!\lfloor} \Omega \\
&\overset{M4}{=} aXW \mathbin{\lfloor\!\lfloor} \Omega + aYW \mathbin{\lfloor\!\lfloor} \Omega \\
&\overset{M3}{=} a(XW \mathbin{\lfloor\!\lfloor} \Omega) + a(YW \mathbin{\lfloor\!\lfloor} \Omega) \\
Q \mathbin{\lfloor\!\lfloor} \Pi &= (aXW + aYW) \mathbin{\lfloor\!\lfloor} \Pi \\
&\overset{M4}{=} aXW \mathbin{\lfloor\!\lfloor} \Pi + aYW \mathbin{\lfloor\!\lfloor} \Pi \\
&\overset{M3}{=} a(XW \mathbin{\lfloor\!\lfloor} \Pi) + a(YW \mathbin{\lfloor\!\lfloor} \Pi) \\
\Omega \mathbin{\lfloor\!\lfloor} \Pi &= bG \mathbin{\lfloor\!\lfloor} \Pi \\
&\overset{M3}{=} b(G \mathbin{\lfloor\!\lfloor} \Pi)
\end{aligned}
$$

All expressions above are still guarded recursive equations in restricted Greibach normal form.
Since the left-merge operation $\mathbin{\lfloor\!\lfloor}$ is closed, it follows from M1 in Table 3 that $\|$ is closed too. □

4.4 MDLe equivalence is decidable

Systems of guarded recursive equations enjoy nice properties in the sense that verifying the bisimulation equivalence is decidable [22].

Theorem 1 ([22])**.** *Let S_1, S_2 be normed systems of guarded recursion equations (over basic process algebras) in restricted Greibach normal form. Then the bisimulation relation $\approx$, that is whether $S_1 \approx S_1$, is decidable.*

Theorem 1 allows us to conclude that

Corollary 1. *If* MDLe*s are written in the form of a system of guarded recursive equations in Greibach normal form over a* BPA*, the bisimulation relation is decidable.*

Proof. Using Lemma 1, each MDLe is written as a context-free language in Greibach normal form. Lemma 2 translates this representation into a system of guarded recursive equations in restricted Greibach normal form over a BPA. By Theorem 1 of [22], language equivalence for systems in (guarded) restricted Greibach normal form such as the MDLes translated using Lemma 2, is decidable up to bisimilarity. □

The following section ensures that bisimilarity is not lost as a result of the introduction of operators $\|$ and $\mathbin{\lfloor\!\lfloor}$.

4.5 MDLe composition preserves bisimilarity

Proposition 1. *The composition operator $\|$ preserves bisimilarity. That is, if $P \approx Q$, then $P\|R \approx Q\|R$.*

Proof. Consider a relation $\mathscr{R}$ over the set of processes, such that $P\|R$ and $Q\|R$ belong to $\mathscr{R}$ whenever $P \approx Q$. We show that $\mathscr{R}$ is a bisimulation.

Case 1. Process P (or Q) executes action a. If $P \approx Q$, then $(P\|R, Q\|R) \in \mathscr{R}$. Assume that $P \xrightarrow{a} P'$. Then by action relation R6 in Table 4, we have $P \xrightarrow{a} P' \Rightarrow P\|R \xrightarrow{a} P'\|R$. Since $P \approx Q$, there exists Q' such that $Q \xrightarrow{a} Q'$, and $P' \approx Q'$. By definition, $(P'\|R, Q'\|R) \in \mathscr{R}$. Similarly, it can be shown that if $Q \xrightarrow{a} Q'$, then there exists a P', with $P' \approx Q'$ and $(P'\|R, Q'\|R) \in \mathscr{R}$.

Case 2. Process R executes action a. Since bisimulation is reflexive, this case reduces to the previous one, and $(P\|R, P\|R) \in \mathscr{R}$.

Case 3. Process P terminates after executing action a ($P \xrightarrow{a} \surd$). Relation R7 of Table 4 implies that $P \xrightarrow{a} \surd \Rightarrow P\|R \xrightarrow{a} R$. Since $P \approx Q$, we need to have $Q \xrightarrow{a} \surd$. Thus, by R7 of Table 4, $Q\|R \xrightarrow{a} R$. By definition, $R \approx R$ and thus the processes derived with the a-transition belong $\mathscr{R}$. The case where Q terminates after executing a is identical.

Case 4. Process R terminates after executing a ($R \xrightarrow{a} \surd$). By R7 of Table 4, $R \xrightarrow{a} \surd \Rightarrow P\|R \xrightarrow{a} P$. Similarly, $R \xrightarrow{a} \surd \Rightarrow Q\|R \xrightarrow{a} Q$. Given that $P \approx Q$, the processes derived from $P\|R$ and $Q\|R$ when R executes a, belong to $\mathscr{R}$.

Case 5. Processes P and R are synchronously execute action a. In this case, we resort to axiom M1 of Table 3, and treat the transitions of P and R separately according to cases 1 and 2 above. The case where Q executes a synchronously with R is identical.

Case 6. Processes P and R terminate synchronously by executing action a. Axiom M1 of Table 3 allows us to treat the synchronous transition to termination as an asynchronous one. In this case, we proceed according to cases 3 and 4.

Thus, for all combinations of possible transitions for $P\|R$ and $Q\|R$, we have that $P\|R \approx Q\|R$ if $P \approx Q$. The conditions of Definition 6 are satisfied and therefore $\mathscr{R}$ is a bisimulation relation. ☐

From Proposition 1 it follows that

Corollary 2. *The composition of* MDLe*s is decidable up to bisimulation equivalence.*

Proof. The operation $'\llfloor'$ is closed (Lemma 3). It follows from M1 in Table 3 that $\|$ is closed too and also preserve bisimilarity (Proposition 1), which means the composition of MDLes can also be written as a system of guarded recursive equations in restricted Greibach normal form over a BPA. By Theorem 1, this composition is decidable. ☐

5 A Case Study: the Sliding Block Puzzle

Representing an instance of the sliding block puzzle as a multi-robot hybrid system serves as a reality check, to ensure that our formulation captures the possible interaction between heterogeneous robot systems. In a general sliding puzzle puzzle, the challenge is to slide blocks on a flat surface with the purpose of achieving a desired configuration. No block can be removed from the board. Quoting Gardiner [27]

> These puzzles are very much in what of a theory. Short of trial and error, no one knows how to determine if a given state is to obtainable from another given state, and if it is obtainable, no one knows how to find the minimum chain of moves for achieving the desired state.

It has been shown that in general, sliding-block puzzles are PSPACE-complete [28, 29]. However, under certain simplifying assumptions and for cases of such puzzles like the one we consider here (Fig. 2), a polynomial algorithm can be constructed to move a single block from any initial position to any final position [29].

In the simple instance of the sliding block puzzle depicted in Fig. 2, the goal is for the robot (initially at position 30) to move the block at position 1 to location 6. Robot and blocks are thought to be autonomous agents, each with its own MDLe. A block can do nothing by itself; any transitions within the block's MDLe may only be activated after composition with the robot agent, which can *push* a block to a different location. However, these potential transitions in the block's configuration need to be encoded in its enabled event set η.

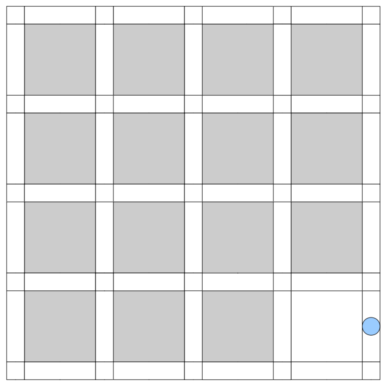

Figure 2: Realization of a sliding block puzzle. Square blocks (tiles) cover all but one cell of a 4 × 4 grid. A robot (round object) is moving along the rows and columns of the grid reconfiguring the blocks. Blocks and robot are modeled as agents moving according to their own MDLe.

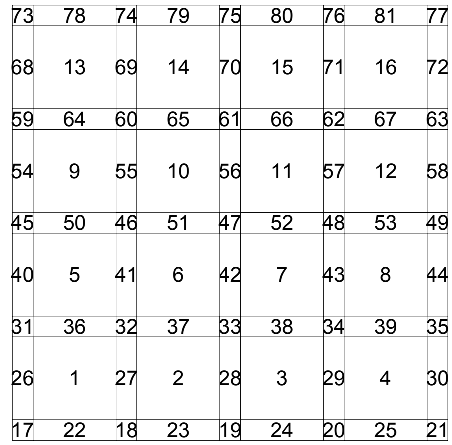

Figure 3: Enumeration of agent positions for the agents in the sliding block puzzle. Positions 1 through 16 can be occupied by blocks. (In Fig. 2, position 4 is not occupied.) Positions 17 through 81 represent possible positions for the robot agent.

For a block to be able to make a transition (which is synchronized with a corresponding on in the robot's event set), the destination location must be unoccupied; thus blocks need to keep track of whether nearby locations are occupied. We therefore model the state of the block as a triplet, consisting of the state of motion (the analogous of the controller in a robotic system), its position, and the availability of an empty location in the immediate neighborhood. The block automaton is $B = (N_b, \eta_b, N_b \cup \eta_b, \Gamma_b, \delta_b, N_{0b}, Z_{0b})$, where

1. $N_b := \{E_{b1}, E_{b2}, E_{b3}\}$ is the set of states, where
   - $E_{b1} \in \{u_1, \dots, u_5\}$ is a motion state: u_1 (be pushed east), u_2 (be pushed west), u_3 (be pushed north), u_4 (be pushed south), u_5 (stay at location);
   - $E_{b2} \in \{1, \dots, 16\}$ is the position of the block; and
   - $E_{b3} \in \{b_1, \dots, b_5\}$ are possible empty nearby locations: b_1 (east), b_2 (west), b_3 (north), b_4 (south), b_5 (none);
2. $\eta_b = \{\nu_b \mid \nu_b = ((u_i, j, b_k), \xi_b)\}$, with i and k in $\{1, \dots 5\}$, and j in $\{1, \dots, 16\}$, includes all events (MDLe atoms; ξ_b is the block's interrupt function) ;
3. $\Gamma_b : N_b \to 2^{\eta_b}$ is the event activation function (initially mapping to $\emptyset$);
4. $\delta_b : N_b \times \eta_b \to N_b$ is the transition function, also mapping to $\emptyset$ since the range of Γ_b is empty, suggesting that the block automaton can make no transitions on its own (except for the case of u_5).

Symbols N_{0b} and Z_{0b} correspond to the initial state and stack symbol, respectively.

For the robot, an atom consists of the state of motion (controller running) and its position. The robot can move along the rows and columns of the grid, and push against a block in order to move it. The automaton for the robot is a tuple $R = (N_r, \eta_r, N_r \cup \eta_r, \Gamma_r, \delta_r, N_{0r}, Z_{0r})$, where

1. $N_r = \{(E_{r1}, E_{r2})\}$ is the set of states, where
   - $E_{r1} \in \{w_1, \dots, w_9\}$ are the available controllers for the robot: w_1 (push east) w_2 (push west), w_3 (push north), w_4 (push south), w_5 (stay at location), w_6 (move east), w_7 (move west), w_8 (move north), w_9 (move south); and
   - $E_{r2} \in \{17, \dots, 81\}$ are the possible positions for the robot;

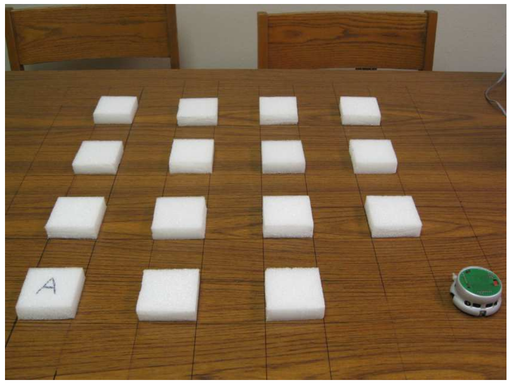

Figure 4: The initial configuration of the robot and blocks.

Figure 5: The final configuration of the robot and blocks.

2. $\eta_r = \{\nu_r \mid \nu_r = ((w_i, j), \xi_r)\}$, where i is in $\{1,\ldots,9\}$, j in $\{17,\ldots,81\}$, and ξ_r is the robot's interrupt function, includes all the events associated with possible robot transitions;

3. $\Gamma_r : N_r \to 2^{\eta_r}$ is the activation function determining which events are active at each robot state; and

4. $\delta_r : N_r \times \eta_r \to N_r$ is the transition function.

Similarly, N_{0r} and Z_{0r} are the initial state and the start stack symbol for the robot automaton, respectively.

The system expressing all possible transitions in the sliding block puzzle is generated by composing the robot with the fifteen blocks. Note that traditional notions of (parallel) composition [26] produce a system where nothing can happen (the puzzle configuration cannot change). However, by identifying "pushing" events in both systems $u_i = w_i$, for $i = 1,\ldots,5$ as common, and including them in $H = \{u_1,\ldots,u_5\}$, the composed system can take synchronized transitions on these events.

According to the distance between these positions, we can define the abstracted atoms as the following (units correspond to encoder counts): Atom0: moving 1032 units; Atom1: turn left; Atom2: moving 4228 units; Atom3: turn right; Atom4: pushing 2044 units; Atom5: moving -2044 units;Atom6: moving 100 units; Atom7: moving 2144 units; Atom8: moving 3176 units. All the plans generated by robot system and block system can be abstracted into the combination of these nine atoms. For example, if the plan for the robot is $(1,26),(5,27),(8,27),(8,32),(8,41),(8,46)$, the abstracted plan will be $(\text{Atom4}),(\text{Atom1}),(\text{Atom8})$.

Figs. 4 and 5 show the position of the robot and the configuration of the block puzzle initially and at the end, respectively. The goal for the robot is to move *blockA* from position 1 to position 6. The plan is the following:

$$(0,2),(1,2),(2,2),(1,2),(0,2),(6,2),(4,2),(5,2),(1,2),(0,2),(1,2),(7,2),(1,2),(0,2),(1,2),(6,2),(4,2),(5,2),$$
$$(1,2),(0,2),(1,2),(7,2),(1,2),(0,2),(1,2),(6,2),(4,2),(5,2),(1,2),(8,2),(3,2),(0,2),(3,2),(6,2),(4,2),(5,2),$$
$$(1,2),(8,2),(3,2),(0,2),(3,2),(6,2),(4,2),(5,2),(1,2),(8,2),(3,2),(0,2),(3,2),(6,2),(4,2)$$

In the plan, the first number represents the abstracted atom index and the second one is the interrupt timeout. The sequence of the motions is shown in Fig. 6. Such plans can be generated automatically using Floyd Warshall's algorithm for shortest paths.

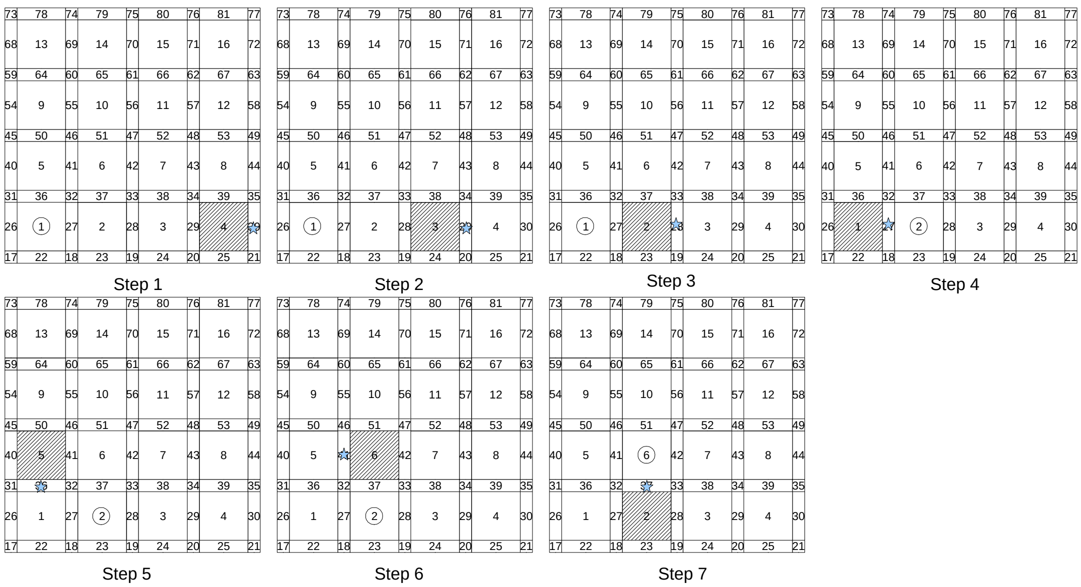

Figure 6: Successive snapshots of the solution to the sliding puzzle problem. The robot ($\star$) moves a block ($\bigcirc$) from position 1 to position 6.

6 Conclusions

Our approach to composition of MDLes and cooperative behavior between heterogeneous systems is based on allowing systems to have additional cooperative transitions, that become active only when the systems are composed with appropriate others. We engineer the mechanics of this interaction by identifying these related, or interdependent, transitions between systems and placing them in a set H that affects how the transitions of the composed system are synchronized. By mapping MDLes to a specific type of basic process algebras we obtain well defined semantics to such compositions, and established computability properties (at least when it comes to language equivalence) for these processes and their compositions.

Acknowledgments

This work is supported by NSF under grant # 0447898. The authors specifically thank Magnus Egerstedt and Rupak Majumdar for insightful discussions.

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[2] G. Fainekos, A. Girard, H. K. Gazit, and G. J. Pappas, "Temporal logic planning for dynamic robots," *Automatica*, 2007, (in press).

[3] V. Manikonda, P. Krishnaprasad, and J. Hendler, "Languages, behaviors, hybrid architectures and motion control," in *Mathematical Control Theory*, J. Baillieul and J. C. Willems, Eds. Springer-Verlag, 1998, pp. 200–226.

[4] D. Hristu, P. Krishnaprasad, S. Anderson, F. Zhang, L.D'Anna, and P. Sodre, "The MDLe engine: A software tool for hybrid motion control," Institute for Systems Research, University of Maryland, Tech. Rep. 2000-54, 2000.

[5] R. Brockett, "Formal languages for motion description and map making," in *Robotics*, J. Bailleul, R. Brockett, and B. Donald, Eds. ACM, 1990, vol. 41, pp. 181–193.

[6] R. W. Brockett, "On the computer control of movement," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1988, pp. 534–540.

[7] C. Stirling, "Decidability of DPDA equivalence," *Theoretical Computer Science*, vol. 255, no. 1–2, pp. 1–31, 2001.

[8] G. Sénizergues, "The equivalence problem for deterministic pushdown automata is decidable," pp. 671–681, 1997.

[9] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. on Robotics*, vol. 21, no. 6, pp. 1077–1091, December 2005.

[10] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[11] P. Jancar, "Undecidability of bisimilarity for petri nets and some related problems," *Theoretical Computer Science*, vol. 148, no. 2, pp. 281–301, 1995.

[12] D. Hristu-Varsakelis, M. Egerstedt, and P. Krishnaparsad, "On the structural complexity of the motion description language MDLe," in *Proceedings of the 42nd IEEE Conference on Descision and Control*, 2003, pp. 3360–3365.

[13] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Sciences. Springer-Verlag, 1980, vol. 42.

[14] ——, *Communication and Concurrency*. Prentice-Hall, 1989.

[15] C. Hoare, *Communicating Sequential Processes*, ser. Lecture Notes in Computer Sciences. Prentice-Hall, 1985.

[16] J. C. M. Baeten and W. P. Weijland, *Process Algebra*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1991, vol. 18.

[17] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL - a tool suite for automatic verification of real-time systems," in *DIMACS Workshop on Verification and Control of Hybrid Systems*. Springer Verlag, 1995.

[18] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "A user guide to HYTECH," in *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science 1019, E. Brinksma, W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, Eds. Springer-Verlag, 1995, vol. 1019, pp. 41–71.

[19] S. Yovine, "KRONOS: A verification tool for real-time systems," *Springer International Journal of Software Tools for Technology Transfer*, vol. 1, no. 1-2, December 1997.

[20] J. Baeten, "A brief history of process algebra," Vakgroep Informatica, Technische Universiteit Eindhoven, Tech. Rep. CSR 04-02, 2004.

[21] Y. Hirshfeld and M. Jerrum, "Bisimulation equivalence is decidable for normed process algebra," School of Informatics at the University of Edinburgh, Tech. Rep. ECS-LFCS-98-386, May 1998.

[22] J. Baeten, J. Bergstra, and J.W.Klop, "Decidability of bisimulation equivalence for process generating context-free languages," *Journal of the ACM*, vol. 40, no. 3, pp. 653–683, 1993.

[23] Y. Hirshfeld, M. Jerrum, and F. Moller, "A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel process," *Theoretical Computer Science*, vol. 158, pp. 143–159, 1996.

[24] G. J. Pappas, "Bisimilar linear systems," *Automatica*, vol. 39, no. 12, pp. 2035–2047, December 2003.

[25] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[26] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic, 2001.

[27] M. Gardner, "The hypnotic fascination of sliding-block puzzles," *Scientific American*, vol. 210, pp. 122–130, 1964.

[28] J. Hopcroft, J. Schwarz, , and M. Sharir, "On the complexity of motion planning for multiple independent objects:pspace-hardness of the 'warehouseman's problem." *International Journal of Robotics Tesearch*, vol. 3, no. 4, pp. 76–88, 1984.

[29] E. D. D. Robert A. Hearn, "Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theoretical Computer Science*, vol. 343, no. 1-2, pp. 72–96, October 2005.